CMSC 447 ⧫ Software Engineering 1 ⧫ Professor Druffel



System Design Document

# Crisis Response Ticket System for Shawn Davis by Team Cloud

**Team Members:** David Foster, Jack McGrann, Julie Nau, Patrick Wheeler

**Last updated:** October 17, 2020

System Design Document

**Table of Contents**

[**1**](#_gjdgxs) **Introduction 2**

[1.1](#_30j0zll) Purpose of This Document 2

[1.2](#_1fob9te) References 3

[**2**](#_3znysh7) **System Architecture 3**

[2.1](#_2et92p0) Architectural Design 3

[2.2](#_tyjcwt) Decomposition Description 3

[**3**](#_3dy6vkm) **Persistent Data Design 5**

[3.1](#_1t3h5sf) Database Descriptions 5

[3.2](#_4d34og8) File Descriptions 5

[3.3](#_2s8eyo1) Requirements Matrix 5

[**4**](#_17dp8vu) **Open Issues 6**

[**5**](#_3rdcrjn) **Appendix A – Agreement Between Customer and Contractor 6**

[**6**](#_26in1rg) **Appendix B – Team Review Sign-off 6**

[**7**](#_lnxbz9) **Appendix C – Document Contributions 6**

**Document Versioning Control**

|  |  |  |  |
| --- | --- | --- | --- |
| Version Number | Date | Changes from Previous Version | Author |
| 1.0 |  | Original Document |  |
|  |  |  |  |

# Introduction

## Purpose of This Document

This document describes in detail the structure and design of the Crisis Response Ticket System, including its general system architecture and persistent database. The document explains terminology relating to its ticketing structure and how said structure is handled across the system’s two layers, followed by graphical diagrams demonstrating the divisions between the system’s UI and data layers. This is then followed by a functional decomposition diagram expanding on the purpose and capabilities of the software. The persistent data storage system is next thoroughly explained, including a description of each data type associated with the database, a description of the database design, and a brief analysis of system components as they relate to the functional requirements of the system. This information is provided in this format in order to ensure consistency and proper functionality of the system.

## References

Druffel, K. (2020, September). 2-ProjectDescription-CalltoMission. Retrieved September 18, 2020, from <https://umbc-my.sharepoint.com/:w:/g/personal/kdruffel_umbc_edu/EYVFEiSwyVJHv5VKzjr2OJUBkPo4klqrugJTDvI-cusDCA?e=SbTP27>

Druffel, K. (2020). 447-10-13-Design-Diagrams [PowerPoint slides]. Retrieved from <https://umbc-my.sharepoint.com/:p:/g/personal/kdruffel_umbc_edu/EdGKWmw3jh1NiKZV6WqIu6cBjW7F80tdi1nB0dZ_5ZhFxQ?e=ydrvRI>

Druffel, K. (2020, September). 447-00-Timeline-20b. Retrieved September 27, 2020, from <https://umbc-my.sharepoint.com/:x:/g/personal/kdruffel_umbc_edu/EU4VoggR0ihEgVqFVYqbfh8BgSj96iz2EQLeNlxRpX0Ebg?e=91dtyj>

Druffel, K. (2020, October). 30-SDD-TeamName-Template-F20. Retrieved October 23, 2020, from <https://docs.google.com/document/d/1qboWoiVLan5KjjCvyh4-czRMhwBocIpY/edit>

Druffel, K. (2020, October). 30-sddGradingRubric\_S18. Retrieved October 23, 2020, from <https://umbc-my.sharepoint.com/:w:/g/personal/kdruffel_umbc_edu/Ee2M2QKci_JGjuuyZkFiqmcBrjmwkFYIeFN1BBsGtRhYtw?e=Gv24sv>

# System Architecture

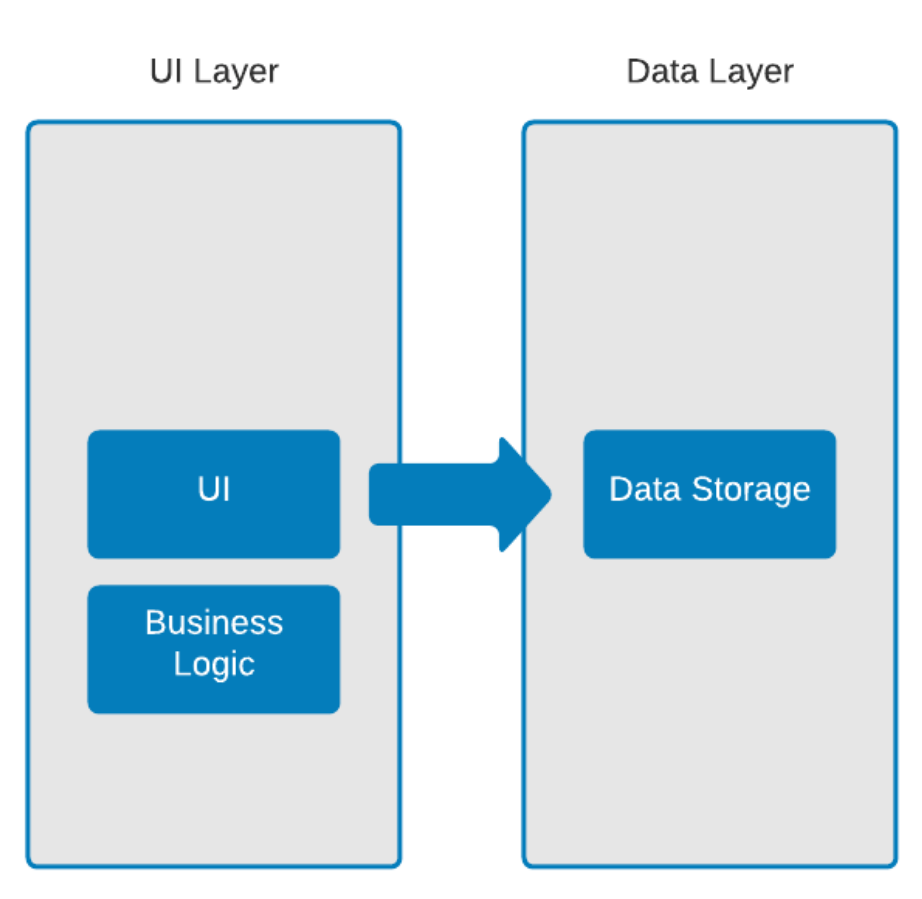
The architecture of this system is built on our data tables. Data enters the system through premade data points to simulate real-life events which are then entered as tickets int othe system databases. These tickets are organized into mission data objects by the Operations Chief. Each user of the system has a specific view of the data based on their user role that displays information from the database and allows them to modify the appropriate parts, which then change the views as appropriate for the other users.

The term UI stands for User Interface which defines the screen that the user sees while using the program. The system is two-layered with the UI Layer (user input, system output) handling all GUI and I/O while the Data Layer (databases, stores info about tickets and missions) handles the database with accompanying databases that serves information to each user’s UI.

## Architectural Design

We chose a two-layer architecture for the simplicity of structure. The presentation layer faces the user, showing them all the necessary information while giving them the tools they need to modify and respond to it. The data requests from the UI Layer are sent to the data layer, where the data is stored on the server. This meets the customer requirements of allowing the users to access and modify event information through the ticket objects (Further explained in section 3.3).

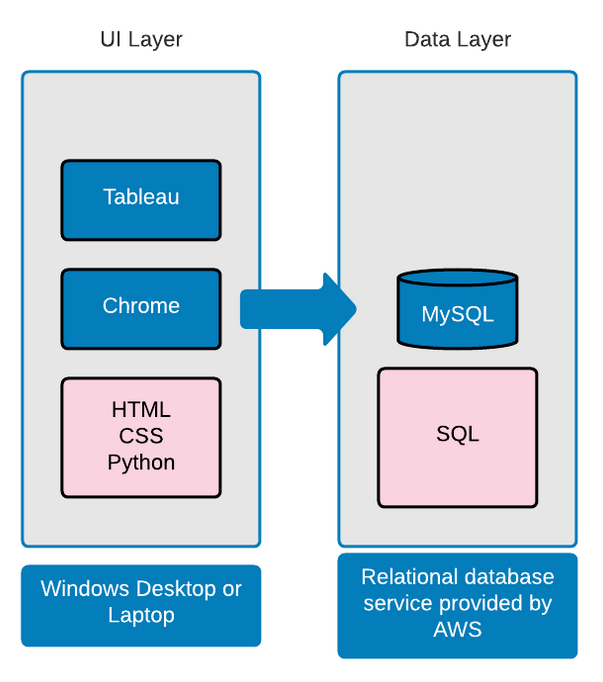
Our team is familiar with the MySQL database system as well as SQL queries and the implementation of SQL with Python, making these tools a great choice for the system architecture.



*Figure 1- Logical Architecture Diagram*

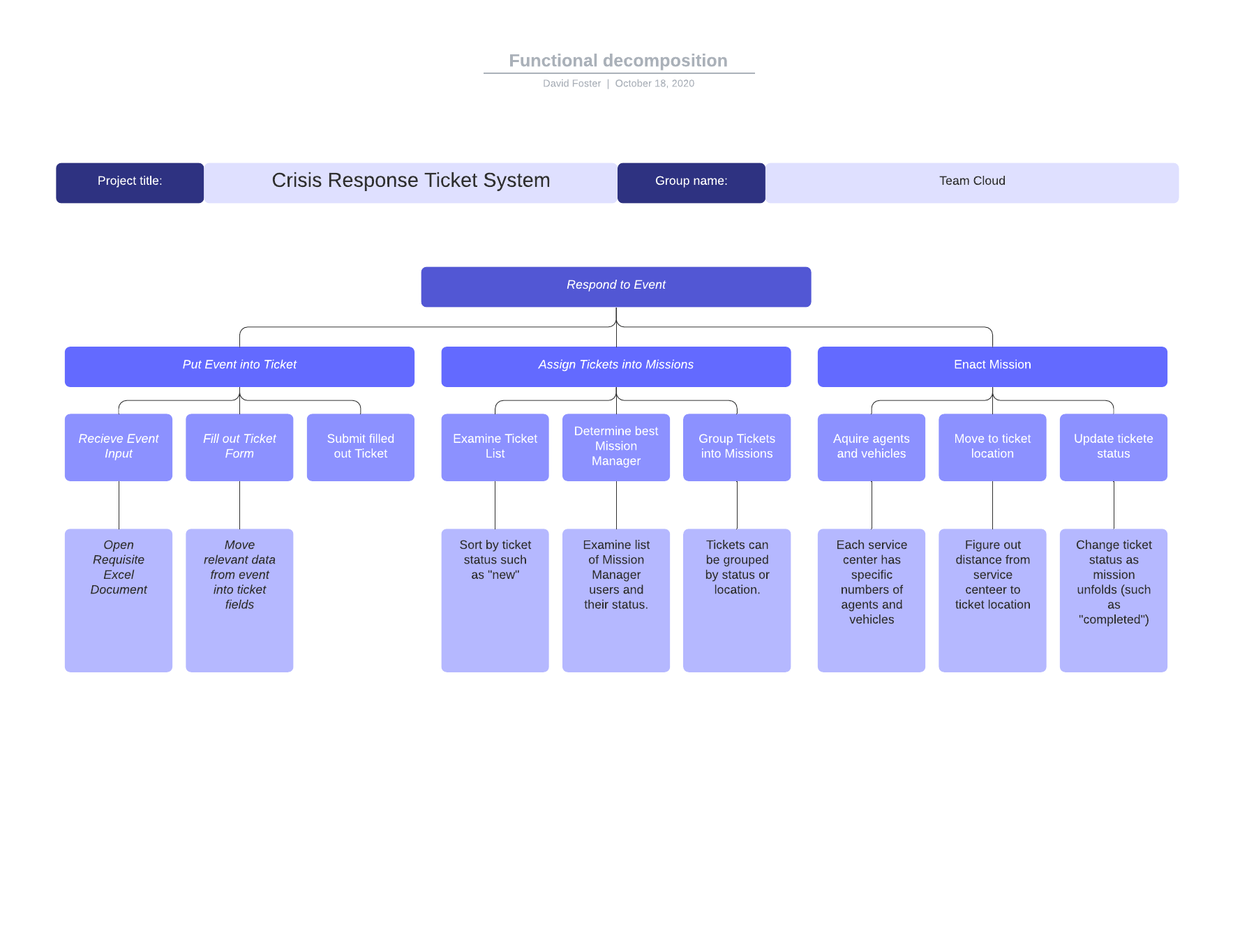
In our Logical Architecture Diagram (Figure 1 above), the service has been divided into two tiers or layers. The first layer is the UI layer that contains the user interface but also the business logic that dictates the rules in which different users can access and modify elements of the UI and the data. When data is modified in the UI layer, the system then communicates to the Data layer to modify the databases. When the databases are modified , the Data layer sends the new information to be displayed to the user on the UI layer.

In a larger-scale application of this program, the data layer would be stored on a dedicated server. For the purposes of this specific application, the data will be hosted using Amazon Web Services with a MySQL database. AWS is a powerful scalable data-hosting service that will allow us to hold as much information as is required by the database at a good cost. The UI will be displayed on a conventional web browser such as Google Chrome accessed through a typical Windows-enabled laptop or desktop system, which can be seen in Figure 2 below.



*Figure 2: Technology Architecture Diagram*

## Decomposition Description

The system as built by the components described above (See Figure 2) will take in the data as crisis events that the Call Center Operator then fills in as data objects to the system known as “tickets”, which become visible to the other users. Operations Chief users can then group these “tickets” into “missions” which become visible to Mission Manager users. This process is laid out below in Figure 3, describing the flow of data through the system beginning at the left side and moving down all the way to the right side. Once a mission is completed, all tickets contained inside have their status changed to “completed”.**

*Figure 3: Functional decomposition*

In Figure 3 above, the primary function of the system is to respond to an event. Responding to an event means the Call Center Operator user taking the crisis event information and constructing a ticket object, assigning the ticket into a mission, then enacting and completing that mission.

In order to put the event into a ticket the Call Center Operator User receives the event input, fills out the ticket form, then submits it into the database.

Once the data has entered the system, the ticket becomes visible as a new ticket to the Operations Chief user. This user can examine the list of tickets, then select a mission manager, and create a new mission assigned to that mission manager with one or more tickets included.

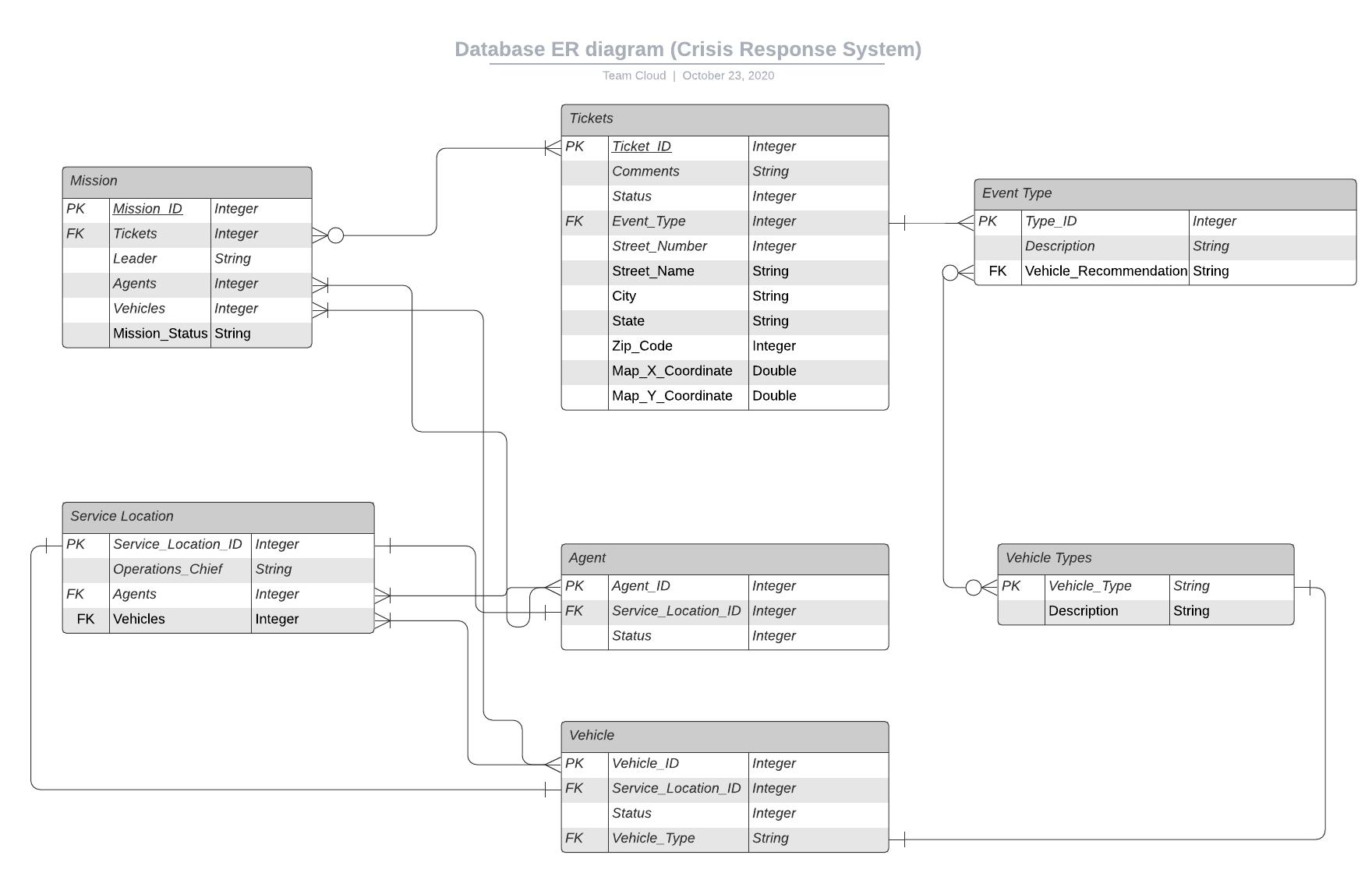
Now that the data has been passed to the Mission Manager user, they can then assign agents and vehicles from the available resources at their service location to dispatch the mission, move to the mission location, and complete the mission. Each of these steps involves changing the mission status, such as “new” to “dispatched” to “en route” to “in-progress” to “completed”.

During all this the system admin will be on hand to revert any mistaken status changes and monitor the user experience for all user types.

# Persistent Data Design

In this section the document will be reviewing all of the necessary details that the reader will need in order to understand how the Crisis Response Ticket System is implemented. This document provides an entity relationship diagram (ERD) in order to give a visual for how the database will be created and interact with itself and then describes in detail the specific components of each part of the database in tabular format. In a similar manner, the document will then provide a diagram for any files that will be created in the process of implementing the system and how they interact with one another as well as give any details necessary for the composition of said files. Finally in this section the document will provide a table showing which system components satisfy each of the functional requirements laid out in the System Requirement Specifications by use case number and name.

## Database Descriptions



*Figure 4*

Table: Tickets

|  |  |  |
| --- | --- | --- |
| **Field** | **Format** | **Description/Comments** |
| Ticket\_ID | Integer | For each event entered into the database, the system will assign an auto-incrementing primary key integer for the Ticket\_ID. |
| Comments | String | This field serves as comment space for any special details agents would need to deal with the event. |
| Status | Integer | This integer will show whether or not this particular ticket is currently assigned to a mission or not (0 for not assigned, 1 for assigned to a mission). |
| Event\_Type | Integer | When new events are entered as tickets, an integer will be chosen to represent what kind of emergency is taking place (eg 1 for fire, 2 for flooding, etc). This value acts as a foreign key to the table with corresponding information detailing the type of event. The user will choose from a list of values showing which emergency type they correspond to and will pick the correct one during ticket creation. |
| Street\_Number | Integer | This integer is the street number in the address where the emergency is taking place, entered during ticket creation by the user. |
| Street\_Name | String | This string is the street name in the address where the emergency is taking place, entered during ticket creation by the user. |
| City | String | The city in the address where the emergency is taking place, entered during ticket creation by the user. |
| State | String | The state in the address where the emergency is taking place, entered during ticket creation by the user. |
| ZIP\_Code | Integer | The ZIP code for the address where the emergency is taking place, entered during ticket creation by the user. |
| Map\_X\_Coordinate | Integer | X-coordinate for where the event is on the map visualization. This will be generated by the mapping API based on the address data provided by the user. |
| Map\_Y\_Coordinate | Integer | Y-coordinate for where the event is on the map visualization. This will be generated by the mapping API based on the address data provided by the user. |

Table: Event Type

|  |  |  |
| --- | --- | --- |
| **Field** | **Format** | **Description/Comments** |
| Type\_ID | Integer | Each emergency event type will have a unique corresponding integer value (eg 1 for fire, 2 for flooding, etc). This value will be the primary key identifier for each type of emergency event listed in this table. |
| Description | String | This string describes which type of emergency event is associated with the Type\_ID (eg “Fire”, “Flooding”, “Electrical”, etc). |
| Vehicle\_Recommendation | String | Gives a list of emergency vehicles in string format that are necessary to handle this emergency type. This acts as a foreign key to the Vehicle Types table which has the types of vehicles that can be listed.The system can recommend no vehicles or many vehicles. |

Table: Service Location

|  |  |  |
| --- | --- | --- |
| **Field** | **Format** | **Description/Comments** |
| Service\_Location\_ID | Integer | Each service location will have an auto-incrementing primary key integer assigned to it when it is input into the system as an available service location. |
| Operations\_Chief | String | This string is the name of the Operations Chief present at this particular service station. |
| Agents | Integer | This is a list of Agent\_ID’s that show which agents in the system are assigned to this service station. These values will act as foreign key identifiers to the list of agents in the system provided by the Agent table. Each service location will have multiple agents. |
| Vehicles | Integer | This is a list of Vehicle\_ID’s that show which particular vehicles in the system are assigned to this service station. These values will act as foreign key identifiers to the list of vehicles in the system provided by the Vehicle table. Each service location will have multiple vehicles. |

Table: Agent

|  |  |  |
| --- | --- | --- |
| **Field** | **Format** | **Description/Comments** |
| Agent\_ID | Integer | Each agent will have an auto-incrementing primary key integer assigned to it whenever an agent is added to the system. |
| Service\_Location\_ID | Integer | This is a foreign key showing which service location each agent is assigned to. Every agent must be assigned to a service location. |
| Status | Integer | This integer will show whether or not this particular agent is currently assigned to a mission or not (0 for not assigned, 1 for assigned to a mission). |

Table: Vehicle

|  |  |  |
| --- | --- | --- |
| **Field** | **Format** | **Description/Comments** |
| Vehicle\_ID | Integer | Every vehicle that is available in the system for use must have an auto-incrementing primary key integer assigned to it when it is added to the system. |
| Service\_Location\_ID | Integer | This is a foreign key showing which service location each vehicle is assigned to. Each vehicle must be assigned to a service location. |
| Status | Integer | This integer will show whether or not this particular vehicle is currently assigned to a mission or not (0 for not assigned, 1 for assigned to mission). |
| Vehicle\_Type | String | This string acts as a foreign key to the Vehicle Types table and states what kind of emergency vehicle this vehicle is. Each vehicle must have a specified vehicle type chosen from the options available in the Vehicle Types table. |

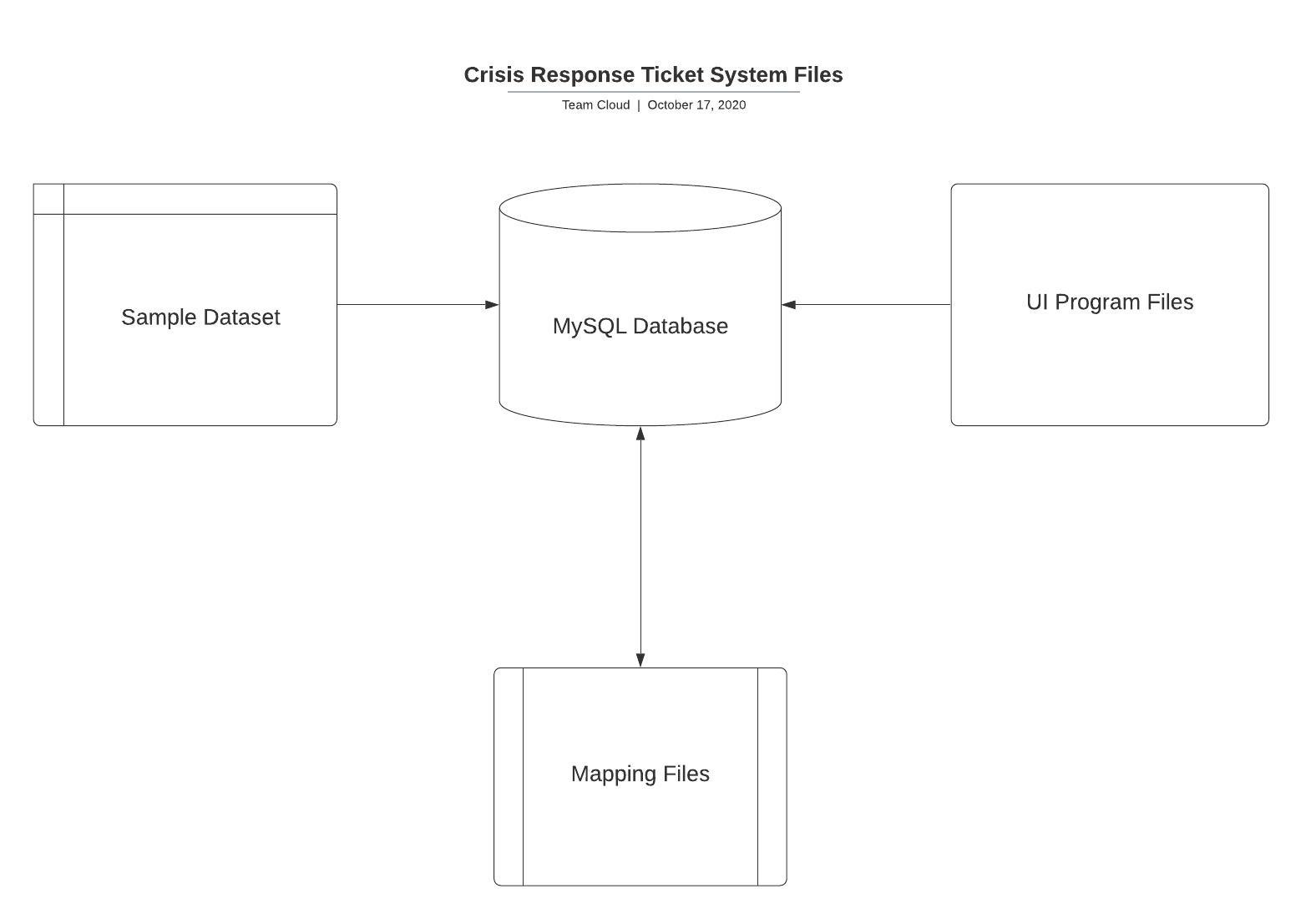
Table: Vehicle Types

|  |  |  |
| --- | --- | --- |
| **Field** | **Format** | **Description/Comments** |
| Vehicle\_Type | String | This string will act as a unique primary key for each type of emergency vehicle provided by the system (eg “firetruck”, “ambulance”, etc). |
| Description | String | Each vehicle type will have a list of strings showing which types of emergency events this type of vehicle can be assigned to (eg “Fire”, “Flooding”, “Electrical”, etc). |

Table: Mission

|  |  |  |
| --- | --- | --- |
| **Field** | **Format** | **Description/Comments** |
| Mission\_ID | Integer | Each mission created by the user will have an auto-incrementing primary key integer assigned to it by the system upon creation. |
| Tickets | Integer | Each mission will have a list of Ticket\_ID foreign keys to show which tickets from the Tickets table will be handled on this mission. The user will input which tickets are assigned to each mission. |
| Leader | String | This string value is the name of the Mission Manager for this particular mission. |
| Agents | Integer | This is a list of Agent\_ID’s that show which agents in the system are assigned to this particular mission. These values will act as foreign key identifiers to the list of agents in the system provided by the Agent table. Each mission will have available agents assigned to it by the Mission Manager upon creation of the mission. |
| Vehicles | Integer | This is a list of Vehicle\_ID’s that show which particular vehicles in the system are assigned to this mission. These values will act as foreign key identifiers to the list of vehicles in the system provided by the Vehicle table. Each mission will have available vehicles assigned to it by the Mission Manager upon creation of the mission. |
| Mission\_Status | String | This string will show the current status of the mission out of four possible mission statuses: “New”, “Assigned”, “In-Progress”, and “Completed”, and will be updated by the mission manager. |

## File Descriptions



*Figure 5*

For the purposes of this project, the system will require a sample dataset in order to populate the Tickets table as it is described above in Section 3.1. This sample dataset will be created on Excel and saved as a csv before being uploaded to the database, and will follow the following format (showing headers for the data we will be manually populating, and the data type it must be to work with the system):

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Comments | Status | Event Type | Street Number | Street Name | City | State | ZIP Code | Map X Coordinate | Map Y Coordinate |
| *String* | *String* | *Integer* | *Integer* | *String* | *String* | *String* | *Integer* | *Double* | *Double* |

Once this csv of sample tickets has been populated with enough samples, the framework of the database diagrammed in section 3.1 will be constructed in MySQL. The csv file holding the sample ticket dataset will be uploaded into the Tickets table on the MySQL database, and each sample ticket will be assigned an auto-incrementing primary key identifier by MySQL. The other tables in the MySQL database will be populated manually, except for the Mission table which will be updated by the user. The database will be hosted on Amazon’s Relational Database Service, provided on Amazon Web Services (AWS). Tableau is a mapping API which will be used to connect an interactive map to the database hosted on AWS. This map will be updated real time to reflect how new tickets and events are added and handled by the system. Tableau saves the workbooks with different map views and data connection files into a shareable Tableau file that can be shared and, once finalized, embedded into our final application. The final set of files that are required to implement the system are the programming files. A combination of CSS and Python files will be used to develop the GUI’s as they have been outlined in the User Interface Design Document.

## Requirements Matrix

|  |  |
| --- | --- |
| Functional Requirement (use case number and name from SRS): | Functions/Methods to satisfy the requirement (From Figure 3: Functional Decomposition):  *All Use Cases will be taking advantage of the GUI’s and the database provided by the system* |
| Use Case 1 - Create Ticket | Receive Event Input  Fill out Ticket Form  Submit filled out Ticket |
| Use Case 2 - Organize Tickets into Missions | Examine Ticket List  Determine Best Mission Manager  Group Tickets into Missions  Map Tickets Using Tableau |
| Use Case 3 - Deploy Mission | Acquire Agents and Vehicles  Move to Ticket Location  Update Ticket Status |
| Use Case 4 - Create User Roles | Fill out New User Form  Submit New User Addition |

# Open Issues

|  |  |  |
| --- | --- | --- |
| Issue # | Issue Description | Target Resolution Date |
| 1 | Need to establish which programming languages we will be using in order to implement the GUI’s. Once we have these narrowed down we can establish which tools allow us to connect to the database using those languages, and specify these in Section 3.2. | 11/3/20 |
| 2 | Need to figure out how big of a real world area will be represented on the map. | 11/3/20 |
| 3 | Need to discuss how sample data will be generated | 11/3/20 |

# 

# Appendix A – Agreement Between Customer and Contractor

In signing this document all parties agree that all information presented herein is correct to the best of their knowledge and is according to the desires and intentions of all parties involved.

Should any changes be made to this document in future, a report of those changes will be made available to all parties for consideration during weekly meetings. Outdated versions of this document will be preserved for future reference.

|  |  |  |
| --- | --- | --- |
| **Typed Name** | **Electronic Signature** | **Date** |
| David F | David Foster | 10/25/2020 |
| John M |  |  |
| Julia N | Julia Nau | 10/23/20 |
| Patrick W | Patrick Wheeler | 10/24/2020 |
| Shawn D |  |  |

**Customer comments here:**

# Appendix B – Team Review Sign-off

In signing below, all team members agree that they have reviewed this document and have approved its current version.

|  |  |  |
| --- | --- | --- |
| **Typed Name** | **Electronic Signature** | **Date** |
| David F | David Foster | 10/25/2020 |
| John M | John McGrann | 10/25/2020 |
| Julia N | Julia Nau | 10/23/20 |
| Patrick W | Patrick Wheeler | 10/24/2020 |
| Shawn D |  |  |

**Team comments here:**

# Appendix C – Document Contributions

|  |  |  |
| --- | --- | --- |
| **Team Member** | **Contribution** | **Estimated Percent Work** |
| David F | Cover page, Section 2, 7 | 30% |
| Jack M | Sections 3 and 4 | 20% |
| Julie N | Sections 1, 5, 6 | 20% |
| Patrick W | Team Support | 10% |